Datenbankanbindung mit PHP und MySQL

Inhalt **Seite**

Eine Ausarbeitung von: Sebastian Brandt; ITA40

* Der Kommentarbereich ist ein Beispiel für die Verbindung von PHP mit einer MySQL-Datenbank. Der Schwerpunkt liegt bei der Erklärung der einzelnen relevanten PHP-Funktionen; das Design sollte jeder individuell festlegen und wird hierbei nicht beachtet. Der Quellcode ist auch in diesem Dokument vorhanden und würde sonst zu lang.

1. Vorarbeit

Eine MySQL-Datenbank erstellen

Voraussetzung: Ich setze bei folgender Erklärung voraus, dass bereits ein funktionierendes System mit Apache, PHP und MySQL aufgesetzt wurde.

Eine MySQL Datenbank wird normalerweise, wie Oracle oder andere DBMS (Datenbank-Management-System) auch, über die Konsole administriert. Für Einsteiger empfiehlt es sich, ein Tool wie phpMyAdmin zu verwenden; die aktuellste Version von phpMyAdmin steht auf der Entwicklerseite phpmyadmin.net zum Download zur Verfügung. Komplettpakete wie das XAMPP-Paket von apachefriends.org liefern phpMyAdmin bereits mit.

Als MySQL-User „root“ und dem festgelegten Passwort (dieser hat mit dem gleichnamigen UNIX-Benutzerkonto nichts zu tun), meldet man sich per phpMyAdmin als Datenbankadministrator mit allen Rechten an. Dort kann man nun eine neue Datenbank anlegen. Alternativ geht das mit dem Befehl *mysql_create_db*, was jedoch nicht empfehlenswert ist, da der Benutzer dann relativ viele Rechte hat. Ich werde daher auf diesen Befehl nicht weiter eingehen; die Syntax finden Sie auf php.net.

Einen Datenbankbenutzer anlegen

Aus Sicherheitsgründen sollte für jede Datenbank, sofern möglich, ein separater Benutzer eingerichtet werden, der ausschließlich Zugriff auf diese Datenbank hat. Bringt ein Angreifer eine Datenbank mit ihren Zugangsdaten unter seine Kontrolle, kann er die anderen Datenbanken nicht erreichen. Daher ist es sehr wichtig, den Datenbankbenutzer root niemals für Datenbanken zu verwenden, die man über PHP-Scripte anspricht. Der Angreifer hätte sonst Vollzugriff auf alle Datenbanken.

In phpMyAdmin legt man einen neuen Benutzer im Menü „Rechte“ (auf der Startseite zu finden) an. Die Schritte werden hier nicht einzeln erläutert, da phpMyAdmin in diesem Fall selbsterklärend ist; als Host kann „localhost“ angegeben werden, sofern der Webserver mit PHP-Interpreter, welcher die PHP-Scripte interpretiert, auf der selben Maschine läuft, wie der MySQL-Datenbankserver.

Nachdem man einen Benutzer und eine Datenbank erstellt hat, muss man ihm datenbankspezifische Rechte zuweisen (gleichnamige Option in phpMyAdmin beim Editieren der Rechte eines Benutzers). Der Benutzer sollte mindestens die Rechte SELECT, INSERT, UPDATE und DELETE bekommen, um Einträge auswerten, anlegen, modifizieren und löschen zu können. Das CREATE-Recht sichert dem Benutzer das Recht zum Anlegen von Datenbankobjekten, beispielsweise einer Tabelle, zu. In den meisten Fällen vergibt man neuen Benutzern die Daten- und Strukturrechte an einer Datenbank (gleichnamige Bezeichnungen in phpMyAdmin).

2. Grundlagen

Eine Verbindung mit der Datenbank herstellen

Zuerst muss immer eine Verbindung mit der Datenbank hergestellt werden. In PHP geschieht dies mit der Funktion *mysql_connect*. Dieser Funktion werden in der Regel drei Parameter übergeben: der Host, der Datenbankbenutzer und das Passwort. Es folgt ein Beispiel; in allen Beispielen dieser Dokumentation gehe ich davon aus, das Datenbank- und Webserver auf derselben Maschine laufen, wodurch sich als Host „localhost“ ergibt.

```
$link=mysql_connect("localhost","myuser","mypass");
```

Diesen Verbindungslink weist man in der Regel immer einer Variablen zu. Das ist nützlich, damit man diese Variable (d.h. die Verbindung) später bei anderen Befehlen verwenden kann.

Um eine Webanwendung komfortabel zu halten, empfehle ich, Variablen zu deklarieren, die man bei Bedarf nur noch ändern muss, z.B. dann, wenn sich ein Passwort ändert oder aufgrund einer Kompromittierung geändert werden muss. Diese Variablen definiert man in einer separaten Datei; es folgt ein Beispiel für die Datei „zugang.php“.

```
<?php
      $host='localhost';
      $user='username';
      $pass='mypass';
      $db='datenbankname';
?>
```

Benötigt man in einem PHP-Script nun Zugang zur Datenbank, reicht es, die soeben erstellte Datei zu inkludieren und die Variablen zu verwenden.

```
<?php
      include("zugang.php");
      $link=mysql_connect($host,$user,$pass);
?>
```

Der Vorteil liegt auf der Hand und wurde oben schon erwähnt. Ändern sich die Zugangsdaten für die MySQL-Datenbank, müssen diese nur noch in einer zentralen Datei (hier „zugang.php“) geändert werden.

Ich habe bereits oben eine Variable $db definiert, welche den Datenbanknamen beinhaltet. Dies wird später bei der Funktion *mysql_select_db* eine Rolle spielen, welche im Kapitel „Data Definition Language – Tabellen anlegen“ erklärt wird.

Zunächst ist es ratsam zu prüfen, ob die Verbindung mit der Datenbank aufgebaut werden konnte; dazu überprüft man den Rückgabewert der Funktion *mysql_connect*, d.h. in unserem Fall den Inhalt der Variablen $link. *mysql_connect* gibt FALSE zurück, sofern die Verbindung mit der Datenbank nicht hergestellt werden konnte.

Folgendes Script überprüft, ob die Verbindung zur Datenbank hergestellt werden konnte. Die Verbindung mit der Datenbank sollte, nachdem die gewünschten Transaktionen ausgeführt wurden, wieder geschlossen werden; dies geschieht mit der Funktion *mysql_close*.

```
<?php
      include(“zugang.php”);
      $link=mysql_connect($host,$user,$pass);
      if (!$link) //mysql_connect hat FALSE zurück gegeben
      {
            echo „Die Verbindung mit der Datenbank konnte nicht
                  hergestellt werden“;
      }
      else
      {     /*Die Verbindung war erfolgreich.
            Jetzt können weitere Befehle auf der Datenbank ausgeführt
            werden. Wurden die gewünschten Aktionen ausgeführt, wird die
            Verbindung wieder geschlossen.*/
            mysql_close($link);
      }
?>
```

Data Definition Language – Tabellen anlegen

Die Kommunikation mit dem Datenbankserver geschieht über die Sprache SQL. Mit der DDL (Data Definition Language) ist es möglich, Objekte in der Datenbank anzulegen; das folgende Beispiel legt eine Tabelle an, die den Benutzernamen und ein Passwort speichern könnte:

```
<?php
      include(“zugang.php”);
      $link=mysql_connect($host,$user,$pass);
      if (!$link) //mysql_connect hat FALSE zurück gegeben
      {
            echo „Die Verbindung mit der Datenbank konnte nicht
                  hergestellt werden“;
      }
      else
      {
            $sql="CREATE TABLE users(id int not null auto_increment
            primary key, name varchar (50), password varchar (50))";
            mysql_db_query($db,$sql,$link);
            mysql_close($link);
      }
?>
```

Die Variable $sql speichert das SQL-Statement; die Funktion *mysql_db_query* erwartet drei Parameter:

- den Datenbanknamen (diesen haben wir bereits in der „zugang.php“ festgelegt)
- das SQL-Statement
- die Verbindungskennung

Mit der Funktion *mysql_db_query* werden SQL-Statements an eine Datenbank geschickt.

Alternativ kann man den Datenbanknamen schon vorher mit der Funktion *mysql_select_db* festlegen und dann die Funktion *mysql_query* verwenden. Das folgende Beispiel verdeutlicht den Zusammenhang:

```
<?php
      include(“zugang.php”);
      $link=mysql_connect($host,$user,$pass);
      if (!$link) //mysql_connect hat FALSE zurück gegeben
      {
            echo „Die Verbindung mit der Datenbank konnte nicht
                  hergestellt werden“;
      }
      else
      {
            $sql="CREATE TABLE users(id int not null auto_increment
            primary key, name varchar (50), password varchar (50))";
            mysql_select_db($db);
            mysql_query($sql,$link);
            mysql_close($link);
      }
?>
```

Hierbei wird die Datenbank $db (in der „zugang.php“ festgelegt) ausgewählt; d.h. alle weiteren Aktionen mit dem Befehl *mysql_query* betreffen dann diese ausgewählte Datenbank. Für welche der beiden Möglichkeiten man sich letztendlich entscheidet ist Geschmackssache; mir persönlich gefällt die letzte Möglichkeit besser.

Der Befehl CREATE TABLE erstellt mit SQL eine Tabelle, in diesem Fall mit dem Namen „users“. Unsere Tabelle verfügt über drei Spalten:

- die ID, welche immer benötigt wird und automatisch vergeben wird (auto_increment) vom Typ integer
- einen String „name“ vom Typ varchar für die Speicherung des Benutzernamens
- einen String „password“ vom Typ varchar für die Speicherung eines verschlüsselten Passwortes

Ich möchte an dieser Stelle keine Abhandelung von SQL geben. Dies macht keinen Sinn, da wir es im Datenbankunterricht eingehend besprochen haben und da ich mich in diesem Dokument hauptsächlich auf das Zusammenspiel von PHP und MySQL konzentrieren möchte. Ich werde daher nur die wichtigsten SQL-Befehle kurz erläutern.

Data Manipulation Language

Einfügen von Datensätzen

Im Folgenden möchte ich kurz die Syntax von SQL-Befehlen zum Einfügen, Ändern, Löschen und Auswerten von Datensätzen zeigen.

Eingefügt werden Daten mit dem Befehl INSERT INTO in eine Tabelle; Beispiel:

INSERT INTO users (name, password) VALUES ('admin', 'd82k9KD2s8kW');

Ein PHP-Script würde hierbei natürlich das Passwort verschlüsseln (z.B. mit der Funktion *md5*) und Variablen für den Usernamen und das Passwort verwenden.

Ändern von Datensätzen

Geändert werden Datensätze durch SQL mit dem Befehl UPDATE; das folgende Beispiel verdeutlicht diesen Zusammenhang:

```
UPDATE users SET name='administrator' WHERE id=1;
```

Unser soeben angelegter Benutzer wird nun umbenannt: SET attribut = neuer wert. Ohne die WHERE-Bedingung würde der neue Wert für alle Datensätzen übernommen.

Löschen von Datensätzen

Der SQL-Befehl DELETE FROM löscht Datensätzen aus einer Tabelle; Beispiel:

```
DELETE FROM users WHERE id=1;
```

Hiermit wird aus der Tabelle „users“ der zuvor angelegte Benutzer gelöscht. Ohne die WHERE-Bedingung würden alle Datensätze aus der Tabelle gelöscht.

Auswerten von Datensätzen

Um Daten aus einer Tabelle abzufragen, wird der SQL-Befehl SELECT benötigt. Das folgende SQL-Statement liefert alle Datensätze aus der Tabelle users zurück, es werden nur die Spalten „name“ und „password“ selektiert:

```
SELECT name,password FROM users;
```

Mit dem Schlüsselwort FROM wird die Tabelle ausgewählt; durch WHERE Bedingungen können SELECT Abfragen weiter eingegrenzt werden.

3. Ein Kommentarbereich mit PHP und MySQL-Datenbank

An einem Kommentarbereich (eigentlich identisch mit einem Gästebuch) möchte ich das Zusammenspiel von PHP und MySQL genauer erläutern. Dabei werde ich auch Funktionen vorstellen, die das komfortable Auslesen von Datensätzen aus der Datenbank ermöglichen.

Ich setze hierbei voraus, dass bereits ein Datenbankbenutzer sowie eine Datenbank erstellt wurden; der Benutzer benötigt Daten- und Strukturrechte.

Einrichten der Datenbank

Zuerst muss der Aufbau der Datenbank geplant werden, da dieser später nur schwer geändert werden kann; es müssten dann nämlich alle Scripte angepasst werden. Ich möchte in meinem Kommentarbereich neben dem Namen des Kommentators und

seinem Text auch die IP-Adresse speichern. Dies erledigt in PHP die Funktion *getenv*; folgender Codeausschnitt gibt die IP-Adresse aus:

```
<?php
      $ip=getenv("REMOTE_ADDR");
      echo $ip;
?>
```

Zur Speicherung benötigen wir also vier Spalten:
- die ID (Primärschlüssel); immer notwendig
- den Namen des Kommentators
- die Nachricht
- die IP-Adresse

Um das lästige Anlegen mit phpMyAdmin zu vermeiden, schreiben wir zum Anlegen der Tabelle ein kleine Script, „install.php“:

```
<?php
      include("zugang.php");
      $link=mysql_connect($host,$user,$pass);
      if (!$link)
      {
            echo "Die Verbindung mit der Datenbank konnte nicht
                  hergestellt werden";
      }
      else
      {
            $sql="CREATE TABLE comments(id int not null auto_increment
                  primary key, name varchar (50), text longtext not null,
                  ip varchar(20))";
            mysql_select_db($db);
            mysql_query($sql,$link);
            mysql_close($link);
      }
?>
```

Mit dem Typ „longtext“ hat der Benutzer genug Platz für seinen Kommentar; bei den anderen Spalten kann eine Begrenzung der Zeichen angegeben werden: varchar(maximale_anzahl_der_zeichen).

Eingabeformular für den Endbenutzer – Einfügen der Daten in die Datenbank

Nach der soeben erledigten Vorarbeit muss ein Eingabeformular entworfen werden, im dem der Endbenutzer einen Kommentar abgeben kann. Die Daten werden anschließend in der Datenbank gespeichert.

Es folgt mein Lösungsvorschlag („form.php“ auf der Diskette):

```
<html>

<head><title>Kommentar abgeben</title></head>

<body>

<table>

<form action="<?php echo $PHP_SELF; ?>" method="post">

<tr>
<td>Name:</td>
<td><input name="name" type="text" maxlength="20"></td>
</tr>
```

```
<tr>
<td>Kommentar:</td>
<td><textarea name="comment"></textarea></td>
</tr>

</table>

<input name="send" type="submit" value="Kommentar absenden">

</form>

<br><br><a href="show.php">Kommentare ansehen</a>

<?php
  $insert=true;
  if ($_POST[send])
  {
      if (!$_POST[name])
         {
            echo '<br><br><font face="Tahoma" size="2"
            color="cc0000">Bitte geben Sie Ihren Namen ein.</font><br>';
            $insert=false;
         }
         if (!$_POST[comment])
         {
            echo '<br><font face="Tahoma" size="2" color="cc0000">Bitte
            geben Sie einen Kommentartext ein.</font><br>';
            $insert=false;
         }
         if ($insert==true) //alles wurde ausgefuellt
         {
            include("zugang.php");
            $link=mysql_connect($host,$user,$pass);
            $ip=getenv("REMOTE_ADDR"); //Client-IP-Adresse
            if (!$link)
            {
                  echo "Die Verbindung mit der Datenbank konnte nicht
                  hergestellt werden";
            }
            else
            {
                  $sql="INSERT INTO comments (name,text,ip) VALUES
                  ('$_POST[name]','$_POST[comment]','$ip')";
                  mysql_select_db($db);
                  if (mysql_query($sql,$link))
                  {
                        echo '<meta http-equiv="refresh" content="0;
                        URL=show.php">';
                  }
                  else
                  {
                        echo "<br><br>Beim Eintragen der Daten ist ein
                        Fehler aufgetreten.";
                  }
                  mysql_close($link);
            }
       } //Ende der if-Bedingung $insert==true
  }
?>

</body>
</html>
```

Erklärung:

Im <form>-Teil der Datei wird ein Formular erstellt. Wichtig ist es hierbei, das „name“-Attribut zu verwenden, da die Inhalte später unter diesem Variablennamen per POST übertragen werden.

Das „action“-Attribut des form-Tags hat den Wert: `<?php echo $PHP_SELF; ?>`

Die Variable $PHP_SELF beinhaltet immer den Dateinamen; d.h. das HTML-Formular ruft beim Abschicken des Formulars dieselbe Datei noch mal auf. Alternativ hätte man für das Attribut „action“ auch den Wert manuell eintragen können. Der Nachteil ist jedoch, dass man ihn dann, sofern die Datei später umbenannt wird, wieder anpassen muss.

Im PHP-Teil der Datei wird zunächst eine Variable $insert vom Typ bool deklariert und bekommt den Wert TRUE zugewiesen. Diese Variable dient zur Fehlerüberprüfung; der Sinn wird gleich noch deutlich. Da das Formular per POST abgeschickt wurde, befinden sich die Variablen nun im Array: $_POST. Mit „send“ habe ich den Submit-Button benannt; wenn dieser gedrückt wurde (d.h. wenn das Formular abgeschickt wurde), wird die nachfolgende if-Anweisung ausgeführt.

```
if ($_POST[send]) { … }
```

Zunächst wird geprüft, ob Name und Kommentar eingegeben wurden. Falls nicht, bekommt der Benutzer eine entsprechende Fehlermeldung und die Variable $insert wird auf FALSE gesetzt. Die darauf folgende if-Anweisung wird nur ausgeführt, wenn die Variable $insert den Wert TRUE hat; dies ist der Fall, sofern keine Fehlermeldung kam, d.h. wenn sowohl ein Name als auch ein Kommentar eingegeben wurde.

Sofern eine Verbindung zur Datenbank aufgebaut werden kann, wird ein INSERT ausgeführt, d.h. die Daten werden in die Datenbank eingetragen.

```
if (mysql_query($sql,$link))
```

Hierbei wird der Rückgabewert von *mysql_query* geprüft. Ist dieser TRUE (dies ist der Fall, wenn die Daten erfolgreich in die Datenbank eingetragen wurden), wird der Benutzer durch einen meta-Redirect auf die Datei „show.php“ umgeleitet. Diese Datei soll später die Kommentare anzeigen. Ist der Rückgabewert FALSE, erscheint eine entsprechende Fehlermeldung auf der Eingabeseite.

Anzeigen der Kommentare – Daten aus der MySQL-Datenbank abfragen

Auf der Seite „show.php“ sollen die Kommentare angezeigt werden. Auch hierbei möchte ich zuerst meinen Quelltext vorstellen und diesen dann erläutern.

```
<html>
<head><title>Kommentare</title></head>
<body>
<a href="form.php">Kommentar abgeben</a><br><br>

<?php
  include ("zugang.php");
  $link=mysql_connect($host,$user,$pass);
  if (!$link)
  {
      echo "Die Verbindung mit der Datenbank konnte nicht hergestellt
            werden";
  }
```

```
  else
  {
      $sql="SELECT name,text FROM comments";
      mysql_select_db($db);
      $erg=mysql_query($sql,$link);
      for ($i=0; $i<mysql_num_rows($erg); $i++)
      {
            $entry[$i]=mysql_fetch_array($erg);
      }
      mysql_free_result($erg);
      for ($i=0; $i<count($entry); $i++)
      {
            echo $entry[$i][name]."<br>".$entry[$i][text]."<br><br>";
      }
      mysql_close($link);
  }
?>

</body>
</html>
```

Erklärung

```
$erg=mysql_query($sql,$link);
```

Zuerst wird das Ergebnis der Abfrage (SELECT name,text FROM comments) in der Variablen $erg gespeichert.

Um die Datensätze einzeln zu erhalten (Zeile für Zeile), muss die Ergebnismenge (gespeichert in $erg) in ein mehrdimensionales Array geschrieben werden. Dies geschieht hier:

```
for ($i=0; $i<mysql_num_rows($erg); $i++)
{
      $entry[$i]=mysql_fetch_array($erg);
}
```

Die Funktion *mysql_num_rows* ermittelt die Anzahl der Zeilen, die die Abfrage zurückgegeben hat, d.h. die Anzahl der Datensätze die in der Variablen $erg gespeichert sind. Dadurch wird die Anzahl der Schleifendurchläufe festgelegt.

mysql_fetch_array schreibt die Ergebnisliste aus der Variablen $erg in das mehrdimensionale Array $entry. Es handelt sich dabei um ein assoziatives Array; die Feldnamen entsprechen den Spaltennamen in der Datenbanktabelle (in diesem Fall „name“ und „text“).

```
for ($i=0; $i<count($entry); $i++)
{
      echo $entry[$i][name]."<br>".$entry[$i][text]."<br><br>";
}
```

Die Ausgabe gestaltet sich somit sehr komfortabel; es werden alle Kommentare ausgegeben. Die Funktion *count* ermittelt die Anzahl der Elemente des Arrays, womit die Schleifendurchläufe festgelegt werden.

```
mysql_free_result($erg);
```

Die Funktion *mysql_free_result* gibt den Arbeitsspeicher, den die Ergebnisliste belegt, wieder frei.

Unterrichtsaufgabe
Administrationsoberfläche zum Löschen von Einträgen

Folgende Aufgabe könnte im Unterricht zu diesem Thema gestellt werden, sofern den Schülern dieses Dokument bekannt ist und/oder sie durch einen Vortrag über den Kommentarbereich informiert wurden.

1. *Schreiben Sie zu dem Kommentarbereich eine Administrationsoberfläche, welche das Löschen beliebiger Einträge ermöglicht.*
2. *Die Eingaben des Benutzers beim Kommentarbereich werden nicht überprüft. Welche Sicherheitsrisiken entstehen dadurch?*
3. *Prüfen Sie die Eingaben des Benutzers, um das Script sicherer zu gestalten!*

Lösungsvorschlag Aufgabe 1 („admin.php“ auf der Diskette)

```
<html>
<head><title>Administration - Kommentare</title></head>
<body>

<?php
  include ("zugang.php");
  $link=mysql_connect($host,$user,$pass);
  if (!$link)
  {
      echo "Die Verbindung mit der Datenbank konnte nicht hergestellt
            werden";
  }
  else
  {
      mysql_select_db($db);
      if (!$_GET[id])
      {
            $sql="SELECT id,name,text FROM comments";
            $erg=mysql_query($sql,$link);
            for ($i=0; $i<mysql_num_rows($erg); $i++)
            {
                  $entry[$i]=mysql_fetch_array($erg);
            }
            mysql_free_result($erg);
            for ($i=0; $i<count($entry); $i++)
            {
                  echo $entry[$i][name]."<br>".$entry[$i][text]."<br>";
                  echo '<a href="admin.php?id='.$entry[$i][id]
                        .'">Löschen</a>';
                  echo "<br><br>";
            }
      }
      else //$_GET[id] gesetzt
      {
            $sql="DELETE FROM comments WHERE id=".$_GET[id];
            if (mysql_query($sql,$link))
            {  //Seite neu laden
                  echo '<meta http-equiv="refresh" content="0;
                        URL=admin.php">';
            }
```

```
          else
          {
             echo "<br><br>Beim Löschen des Eintrags ist ein Fehler
                   aufgetreten.";
          }
     }
     mysql_close($link);
 }
?>

</body>
</html>
```

Erklärung:

Das Script verhält sich ähnlich wie das schon vorgestellte Script „show.php“ – es zeigt, sofern per GET keine Variable „id“ übergeben wird, alle Einträge an.

Neu ist dabei folgende Zeile:

```
echo '<a href="admin.php?id='.$entry[$i][id].'">Löschen</a>';
```

Beim Klick auf den Link “Löschen” soll die gleiche Seite noch mal aufgerufen werden; allerdings wird diesmal per GET die ID des entsprechenden Datensatzes übergeben.

Folglich wird der else-Block der if-Anweisung (Überprüfung, ob die $_GET[id] gesetzt ist) ausgeführt. Zum Löschen des Datensatzes wird folgendes SQL-Statement benötigt:

```
$sql="DELETE FROM comments WHERE id=".$_GET[id];
```

Die Administrationsoberfläche muss vor unberechtigtem Zugriff geschützt werden (beispielsweise kann ein Verzeichnisschutz durch .htaccess realisiert werden). Ansonsten kann jeder durch einfache Anpassung der URL Datensätze löschen.

```
if (mysql_query($sql,$link))
{
      echo '<meta http-equiv="refresh" content="0;URL=admin.php">';
}
```

Diese Anweisung prüft, ob der Datensatz gelöscht werden konnte; sofern dies der Fall war, wird die Seite „admin.php“ neu geladen. Dadurch werden die Daten erneut aus der Datenbank abgefragt und der gelöschte Datensatz erscheint nicht mehr in der Administrationsoberfläche.

Mögliche Antworten - Aufgabe 2

Bei dem Kommentarformular („form.php“) werden die Eingaben des Benutzers nicht überprüft. Es ist generell ein großes Sicherheitsrisiko den Eingaben des Benutzers zu vertrauen.

Vorgabe: Rufen Sie die Datei „form.php“ auf und tätigen Sie einen neuen Eintrag; als Kommentar tragen Sie folgenden JavaScript-Code ein:

```
<script>alert("Die Eingaben muessen gefiltert werden");</script>
```

Diese harmlose Meldung soll nur die Möglichkeiten dieser XSS-Sicherheitslücke (Cross-Site-Scripting) demonstrieren. Würde man auf der Seite Cookies einsetzen, wäre es für den Angreifer ein leichtes Spiel, diese zu stehlen und somit an sensible Daten, z.B. Logininformationen, zu kommen.

Filtern der Benutzereingaben – Lösungsvorschlag Aufgabe 3

PHP stellt dem Programmierer unterschiedliche Funktionen bereit, die einem das Filtern von unzulässigen Benutzereingaben erleichtern.

- Die Funktion *strip_tags* filtert HTML-Tags und PHP-Zeichen
- Die Funktion str_replace ermöglicht es, nach „gefährlichen Strings“ zu suchen und diese zu ersetzen

Beispiel:

```
$_POST[comment]=strip_tags($_POST[comment]);
$_POST[name]=strip_tags($_POST[name]);
$ POST[name]=str replace('onClick','',$ POST[name]);
```

str_replace ersetzt in diesem Fall den Teilstring „onClick“ durch nichts (es wird also entfernt). *onClick* gehört zu den Event-Handlern und ermöglichen zum Beispiel den Aufruf von JavaScript in dem <input>-Feld für den Namen.

4. Quellen

Diese Ausarbeitung wurde von mir, Sebastian Brandt, eigenständig verfasst. Zitate aus Büchern oder Internetseiten sind nicht vorhanden.

Zur Informationsbeschaffung dienten mir folgende Quellen:

Buch: „PHP 5 & MySQL 4 – Der leichte Einstieg“ Taschenbuch
Autor: Christine Peyton, Andre Möller
Verlag: Markt + Technik
ISBN: 3-8272-6775-7
Ausgabe: Die aktuellste Auflage nennt sich „PHP 5.1 und MySQL 4.1“ (gefunden über buecher.de); eine Auflagennummer kann ich in meinem Buch nicht finden.

php.net-Dokumentation

http://de2.php.net/manual/de/function.mysql-connect.php
eingesehen am: Mittwoch, 20.12.06, 17:05 Uhr

http://de2.php.net/manual/de/function.mysql-close.php
eingesehen am: Mittwoch, 20.12.06, 17:07 Uhr

http://de2.php.net/manual/de/function.mysql-query.php
eingesehen am: Donnerstag, 21.12.06, 10:12 Uhr

http://de2.php.net/manual/de/function.msql-select-db.php
eingesehen am: Donnerstag, 21.12.06, 10:15 Uhr

http://de2.php.net/manual/de/function.mysql-db-query.php
eingesehen am: Donnerstag, 21.12.06, 10:17 Uhr

http://de2.php.net/manual/de/function.mysql-num-rows.php
eingesehen am: Freitag, 22.12.06, 09:56 Uhr

http://de2.php.net/manual/de/function.mysql-create-db.php
eingesehen am: Mittwoch, 20.12.06, 18:34 Uhr

http://de2.php.net/manual/de/function.mysql-fetch-array.php
eingesehen am: Freitag, 22.12.06, 10:37 Uhr

Informationen zu Event-Handlern:
SELFHTML e.V.; Swen Wacker
Webseite: http://de.selfhtml.org
genaue URL: http://de.selfhtml.org/html/attribute/eventhandler.htm
eingesehen am: Samstag, 23.12.06, 11:07 Uhr